C# 프로그래밍

- 7 주차 (1강)

SEJONG UNIVERSITY

- 튜플

튜플: 명명되지 않은 튜플 (Unnamed Tuple)

- 필드를 담을 수 있는 형식 이름이 없는 구조체
  - ✓ 프로그램 전체에 사용할 형식 선언이 아닌 즉석에서 사용할 복합 데이터 형식
  - ✓ 값 형식으로 생성된 지역을 벗어나면 스택에서 소멸되므로 메모리 사용 부담이 적음
- 명명되지 않은 튜플 (Unnamed Tuple)
  - ✓ 튜플 사용할 때 필드의 이름을 지정하지 않음

var 사용: 컴파일러가 튜플의 모양을 보고 직접 형식을 결정

```
var tuple = (123, 789);
Console.WriteLine($"{tuple.Item1}, {tuple.Item2}");  // 출력결과: 123, 789
```

튜플은 괄호 사이에 두 개 이상의 필드를 지정함으로써 만들어짐

튜플: 명명된 튜플 (Named Tuple)

- 명명된 튜플 (Named Tuple)
  - ✓ "필드명:" 의 형태로 필드의 이름을 지정하여 선언
- 튜플 정의와 반대 형태로 분해 가능
  - ✓ 특정 필드를 무시하기 위해 "_" 를 이용
  - ✓ 튜플 분해를 이용해 여러 변수를 단번에 생성하고 초기화 가능

```
var tuple2 = (Name: "홍길동", Age: 17);    // 분해
Console.WriteLine($"{tuple2.Name}, {tuple2.Age}"); // 출력결과: 홍길동, 17

var (name, _) = tuple2; // Age 필드는 무시
Console.WriteLine($"{name}");   // 출력결과: 홍길동

var (name2, age2) = ("이순신", 34);    // 여러 변수 생성 및 초기화
Console.WriteLine($"{name2}, {age2}"); // 출력결과: 이순신, 34
```

튜플: 두 튜플 사이의 할당

- 명명되지 않은 튜플과 명명된 튜플 사이의 할당
  - ✓ 필드의 수와 형식이 같으면 할당이 가능

```
var unnamed = ("슈퍼맨", 9999); // (string, int)
var named = (Name: "홍길동", Age: 17); // (string, int)

// 명명된 튜플 = 명명되지 않은 튜플
named = unnamed;
Console.WriteLine($"{named.Name}, {named.Age}"); // 출력결과: 슈퍼맨, 9999

named = ("원더우먼", 10000);

// 명명되지 않은 튜플 = 명명된 튜플
unnamed = named;
Console.WriteLine($"{unnamed.Item1}, {unnamed.Item2}"); // 출력결과: 원더우먼, 10000
```

튜플: 위치 패턴 매칭 (Positional Pattern Matching)

- 튜플은 switch 문 또는 switch 식의 분기조건에 활용 가능
  - ✓ 튜플이 분해된 요소의 위치에 따라 값이 일치하는지 판단

```
var alice = (job: "학생", Age: 17);
var discountRate = alice switch
{
    ("학생", int n) when n < 18 => 0.2,  // 학생 & 18세 미만
    ("학생", _) => 0.1,  // 학생 & 18세 이상
    ("일반", int n) when n < 18 => 0.1,   // 일반 & 18세 미만
    ("일반", _) => 0.05, // 일반 & 18세 이상
    _ => 0,
};
Console.WriteLine(discountRate);    // 출력 0.2
```

튜플: 위치 패턴 매칭 예제코드

```
private static double GetDiscountRate(object client)
{
    return client switch
    {
        ("학생", int n) when n < 18 => 0.2,
        ("학생", _) => 0.1,
        ("일반", int n) when n < 18 => 0.1,
        ("일반", _) => 0.05,
        _ => 0,
    };
}
```

```
static void Main(string[] args)
{
    var alice = (job: "학생", age: 17);
    var bob = (job: "학생", age: 23);
    var charlie = (job: "일반", age: 15);
    var dave = (job: "일반", age: 21);

    Console.WriteLine($"alice   : {GetDiscountRate(alice)}");
    Console.WriteLine($"bob     : {GetDiscountRate(bob)}");
    Console.WriteLine($"charlie : {GetDiscountRate(charlie)}");
    Console.WriteLine($"dave    : {GetDiscountRate(dave)}");
}
```

출력 결과:

```
alice   : 0.2
bob     : 0.1
charlie : 0.1
dave    : 0.05
```

- 배열

SEJONG UNIVERSITY

배열 (Array)

- 같은 데이터 형의 다수의 데이터를 한번에 다뤄야 하는 경우
  - ✓ 각각의 데이터에 대한 변수를 선언한다면 코드의 양 증가
- 배열 변수 한 개로 다수의 데이터 관리 가능

다수의 변수 선언

```
int score_1 = 80;
int score_2 = 74;
int score_3 = 81;
int score_4 = 90;
int score_5 = 34;
```

사용형식

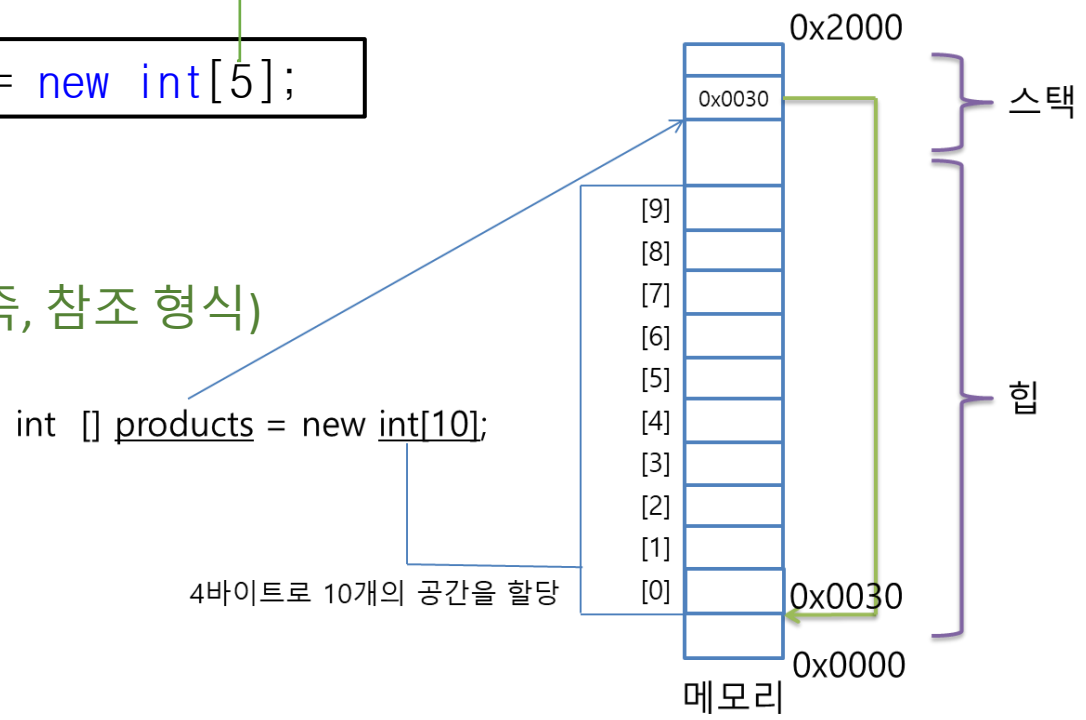
```
데이터형식[] 배열이름 = new 데이터형식[용량];
```

사용예제

```
int[] scores = new int[5];
```

첨자 (index)

배열의 값을 별도로 힙에 할당 (즉, 참조 형식)



배열의 출력과 연산

- 다수의 변수 선언과 배열 사용의 비교

다수의 변수 선언

```
int score_1 = 80;
int score_2 = 74;
int score_3 = 81;
int score_4 = 90;
int score_5 = 34;

Console.WriteLine(score_1);
Console.WriteLine(score_2);
Console.WriteLine(score_3);
Console.WriteLine(score_4);
Console.WriteLine(score_5);

int average = (score_1 + score_2 + score_3 +
                score_4 + score_5) / 5;
```

배열의 사용

```
int[] scores = new int[5];
scores[0] = 80;
scores[1] = 74;
scores[2] = 81;
scores[3] = 90;
scores[4] = 34;

foreach (int score in scores)
    Console.WriteLine(score);

int sum = 0;
foreach (int score in scores)
    sum += score;

int averageArray = sum / scores.Length;
```

Index from end (^) 연산자

- C# 8.0 부터 System.Index 형식과 ^ 연산자 지원
  - ✓ 배열의 마지막부터 역순으로 인덱스를 지정
  - ✓ ^1 은 배열의 마지막 요소를 나타내는 인덱스
  - ✓ ^2 은 배열의 마지막에서 두번째 요소를 나타내는 인덱스
  - ✓ ^ 연산자의 연산 결과는 System.Index 형식의 인스턴스로 나타냄

```
int[] scores = new int[5];
scores[scores.Length - 1] = 34; // scores[4] = 34;와 동일

System.Index last = ^1;
scores[last] = 34;  // scores[scores.Length - 1] = 34;와 동일

scores[^1] = 34;  // scores[scores.Length - 1] = 34;와 동일
```

^ 연산자 예제코드

```
static void Main(string[] args)
{
    int[] scores = new int[5];
    scores[0] = 80;
    scores[1] = 74;
    scores[2] = 81;
    scores[^2] = 90; // 배열 마지막-1
    scores[^1] = 34; // 배열 마지막

    foreach (int score in scores)
        Console.WriteLine(score);

    int sum = 0;
    foreach (int score in scores)
        sum += score;

    int averageArray = sum / scores.Length;
    Console.WriteLine($"Average score: {averageArray}");
}
```

출력 결과:

```
80
74
81
90
34
Average score: 71
```

배열 초기화하는 세 가지 방법

- 배열의 원소 개수 명시, 중괄호 { 와 }로 둘러싸인 블록을 붙인 뒤, 블록 사이에 배열의 각 원소에 입력될 데이터를 입력


```
string[] array1 = new string[3] { "안녕", "Hello", "Halo" };
```

- 첫 번째 방법에서 배열의 용량을 생략

```
string[] array2 = new string[] { "안녕", "Hello", "Halo" };
```

- 첫 번째 방법에서 new 연산자, 형식과 대괄호 [와], 배열의 용량 모두 생략

```
string[] array3 = { "안녕", "Hello", "Halo" };
```

배열 초기화 예제코드

```
static void Main(string[] args)
{
    string[] array1 = new string[3] { "안녕", "Hello", "Halo" };
    Console.WriteLine("array1...");
    foreach (string greeting in array1)
        Console.WriteLine($"{greeting}");

    string[] array2 = new string[] { "안녕", "Hello", "Halo" };
    Console.WriteLine("₩narray2...");
    foreach (string greeting in array2)
        Console.WriteLine($"{greeting}");

    string[] array3 = { "안녕", "Hello", "Halo" };
    Console.WriteLine("₩narray3...");
    foreach (string greeting in array3)
        Console.WriteLine($"{greeting}");
}
```

출력 결과:

```
array1...
안녕
Hello
Halo

array2...
안녕
Hello
Halo

array3...
안녕
Hello
Halo
```

Thank you!